SDMCET Dharwad

Mini-Project Design Report

The patterns used in the project “mpocr”

Project under the guidance of:

- Prof. Sharada H. N

Team

- Madhusoodan Pataki.
- Anarghya Jantali.

Content

Introduction.

The Optical character recognition is the almost solved problem in the computer-world. In this project we are trying to get introduced to the problems that emerge in this way. We try to solve the problem using the Neural-Network. Since this is very first attempt by us we may not get good accuracy but we will try get most out of it. This document describes the way we approached the problem in the SE (Software-Engineering) way. It describes the pattern used to solve the problem, the components of the solution and much more.

Tags

Pattern, Layers, MVC, NeuralNetwork, Neuron, OCR, Training, ActivationFunction, SigmoidFunction, Features, Preprocessing, Segmentation

Plan of Attack.

The first problem we encountered was the decomposition of it. We divided the problem into two parts. It was more like a Layers pattern which distributes the work for each layer. Each layer is further divided into more layers which are explained further.

1. Image processing and Feature extraction.
2. Recognition using Neural Network.

Image processing and Feature extraction.

This step involves the processing of the image which includes binarization, skew detection etc. These all things were implemented in the **`OImage, Segmentation, Segment`** classes which holds the image data along with methods to manipulate it. You can find the methods in UML diagram attached with this document.

Recognition using Neural Network.

This layer classifies the given features into different character classes (i.e recognizes the character). This layer is further subdivided into multiple layers which do same thing as other layers but with different state. They implement a mathematical function which is derived by the training of the network. Each layer takes an fixed length input vector and outputs a fixed length vector which in turn is input to next layer if it's not the last one. The layers are implemented in **`Layer`** class. The data/input passing between layers is carried out by the wrapper class **`NeuralNetwork`** which encapsulates all the **`Layer`** classes. Each **`Layer`** includes several neurons which actually compute the output of the **`Layer`**. The neurons are implemented in **`Neuron`** class. Each neuron takes the weights of the previous layer and computes its activation using the **`IActivationFunction`**'s **`fire`** function.

The components of the layers are explained below.

1. **`BasicImage :`** This is the base class of the all the images used in the solution. It implements basic features such as:
    a. Rotation of the image to the required degree.
    b. Managing the image buffer.

2. **`FeatureSet :`** This class implements the structure of a feature-set which is input to the neural network. It's included in the Segment class.

3. **`IActivationFunction:`** This is an abstract class which asks to define the fire function which computes the activation of a neuron using the weighted input.

4. **`IFeatureSet:`** This is an interface defining the Feature class functionalities.

5. **IImage:** This is the interface which defines the basic requirement to be an Image. The direct implementor of this class is BasicImage. Other extend the BasicImage.

6. **INeuralNetwork:** This interface defines the basic requirement to be a NeuralNetwork. The functions defined here are used by the **NNVisualiser** class to display the **NeuralNetwork.**

7. **Layer:** This class implements sublayers in the recognition layer. It includes the neurons and helps in passing data from previous layer to neuron helping it computing the activations of itself.

8. **MainPage :** It's the driver class of the application it manages the UI along with maintaining the objects such as Neural-Network, Image etc.

9. **Matrix :** This is an util class which implements the basic matrix arithmetics.

10. **mCanvas :** This is somewhat an MVC which helps in displaying current loaded image and state of the image currently.

11. **NeuralNetwork :** This class is the core of recognition layer which manages layers, passes the input between layers and gets the output of them. It also implments the error correction by backpropagation.

12. **Neuron :** Functional core of each layer which computes a part activation of the layer which includes it.

13. **NNVisualizer :** This is a utility class which helps in visulazing the neural network.

14. **OImage :** This is the class derived from **BasicImage** which involves the extended functionalities such as
    a. Converting the image to grayscale.
    b. Binarizing the image
    c. Exporting the image to a file.
    d. Inverting the color of the image.

15. **Segmentation :** This class implements the segmentation which means it divides the image into parts which include the character glyph.

16. **Segment :** This is the part of the image extracted from the OImage from segmentation.

17. **`SigmoidFunction` :** This is an implementation of the **`ActivationFunction`**. The function used here is sigmoid function which is defined as

$$f(t) = \frac{1}{1+e^{-t}}$$

18. **`TextRecognizer` :** Util class which inputs the feature vector extracted from preprocessed image into the neural network and formats the output of the neural-network to the human readable format.

19. **`Trainer` :** Util class to train the Neural Network using the given training-set by preprocessing and extracting features from them.

20. **`Zones` :** A type of feature extracted which derives from **`FeatureSet`** and includes the vector representing the Zones feature.